



Creación de Aplicaciones Python

Lección 2: Conocimientos mínimos generales [2/3]

ÍNDICE

| | |
|---|----------|
| Lección 2. – Conocimientos mínimos generales [2/3] | 2 |
| Presentación y objetivos | 2 |
| 1. Jupyter Notebook: IDE para DATA SCIENCE | 3 |
| 2. PYTHON: Resumen express | 7 |
| 3. Muy breve Introducción al Machine Learning | 19 |
| 4. Puntos claves | 31 |

Lección 2. – Conocimientos mínimos generales [2/3]

PRESENTACIÓN Y OBJETIVOS

Este tema tiene como objetivo cubrir algunos conocimientos necesarios para no “atascarse” a la hora de explicar alguno de los diferentes Frameworks.

Algunas cosas como:

- | Reconocer un Try Except
- | Conocer qué es un Diccionario o un Array como Estructuras de Datos
- | Medio entender la estructura de código en Machine Learning
- | Familiarizarse un poco más con el Lenguaje de Programación Python

La parte de Python será explicada con bastante rapidez dados los conocimientos iniciales de los/as alumnos/as. No obstante, Python es muy amplio, y pudiera haber cosas que no se han aprendido todavía, de modo que explicaremos aquellas cosas necesarias.

Para la parte del *Machine Learning* trabajaremos con un Dataset conocido para poder explicar algunos conocimientos generales. Dicho Dataset será utilizado de una u otra forma posteriormente, en alguno de los Frameworks.



Objetivos

- Repasar brevemente Python.
- Hacer una introducción al Machine Learning.

1. JUPYTER NOTEBOOK: IDE PARA DATA SCIENCE

Para usar el Entorno Jupyter podríamos instalar el Software de Anaconda, el cual nos instalaría más Entornos, no solamente Jupyter.

Ello nos aportaría a su vez más ventajas, entre las cuales, disponer de muchas librerías científicas preinstaladas.

Pero, lo más rápido es instalar Jupyter directamente desde la cmd.

Nos vamos al buscador y escribimos "cmd" para poder instalar nuestro Entorno, lo cual será bastante rápido.



A continuación explicaremos paso a paso el proceso:

Resulta recomendable comenzar viendo qué versión tenemos instalada, no es que sea necesario del todo, pero es algo que puede venir bien.

Ver la versión de Python instalada: "pip install --version"

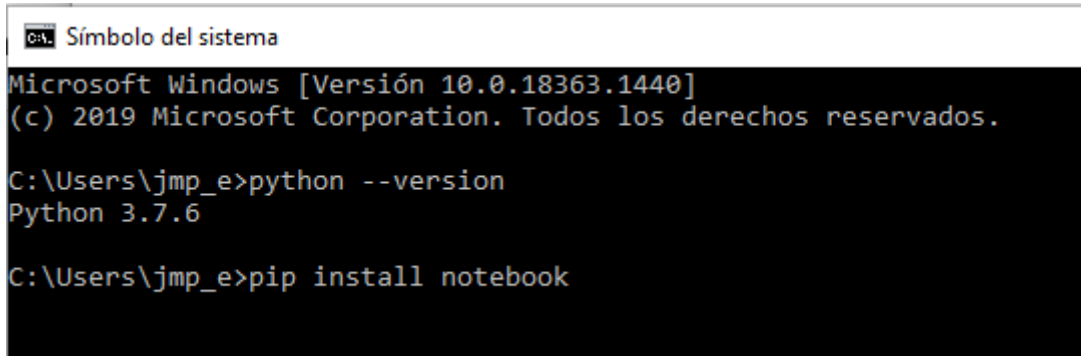
```
cmd Símbolo del sistema
Microsoft Windows [Versión 10.0.18363.1440]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\jmp_e>python --version
Python 3.7.6
```

Figura 1.1: Ver la versión de Python

Tendremos en cuenta qué versiones estables hay disponibles a día de hoy, y cuál es la más reciente, por ejemplo.

Instalar Jupyter Notebook (voluntario): "pip install notebook"



```
C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.18363.1440]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

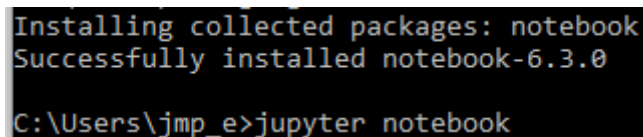
C:\Users\jmp_e>python --version
Python 3.7.6

C:\Users\jmp_e>pip install notebook
```

Figura 1.2: Instalar Jupyter

En menos de 1 minuto se tiene

Abrir el Entorno: "jupyter notebook"

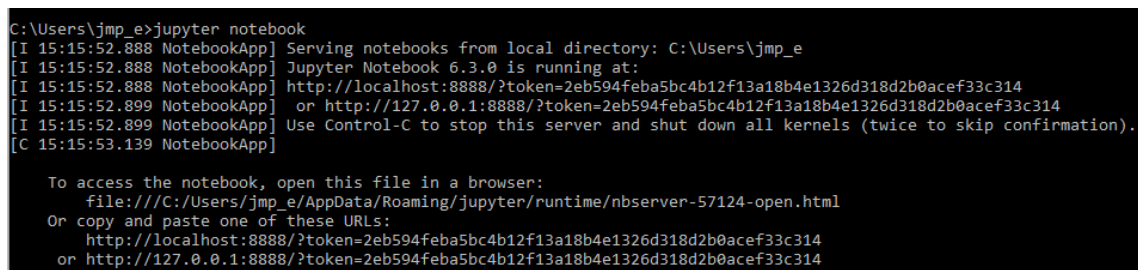


```
Installing collected packages: notebook
Successfully installed notebook-6.3.0

C:\Users\jmp_e>jupyter notebook
```

Figura 1.3: "jupyter notebook" en la cmd

Saldrá algo tal que así.



```
C:\Users\jmp_e>jupyter notebook
[I 15:15:52.888 NotebookApp] Serving notebooks from local directory: C:\Users\jmp_e
[I 15:15:52.888 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 15:15:52.888 NotebookApp] http://localhost:8888/?token=2eb594feba5bc4b12f13a18b4e1326d318d2b0acef33c314
[I 15:15:52.899 NotebookApp] or http://127.0.0.1:8888/?token=2eb594feba5bc4b12f13a18b4e1326d318d2b0acef33c314
[I 15:15:52.899 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:15:53.139 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/jmp_e/AppData/Roaming/jupyter/runtime/nbserver-57124-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=2eb594feba5bc4b12f13a18b4e1326d318d2b0acef33c314
or http://127.0.0.1:8888/?token=2eb594feba5bc4b12f13a18b4e1326d318d2b0acef33c314
```

Figura 1.4: Jupyter en la cmd

Es probable que nos pregunte con que Navegador queremos abrirlo:

Google Chrome, por ejemplo.



Figura 1.5: Entorno Jupyter

Crear un nuevo Python 3 y renombrarlo

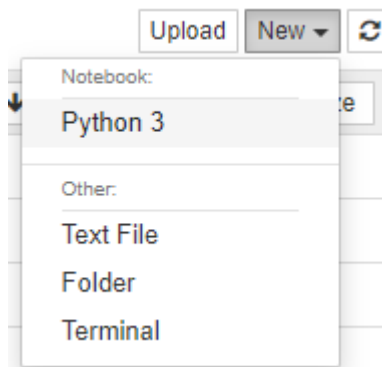


Figura 1.6: Abrir un nuevo Python 3 en Jupyter

Hacemos click en "Untitled" y podemos cambiar el título. "Rename" después de ello.

Nota: En mi caso "Untitled5" porque probablemente tendré algunos más.

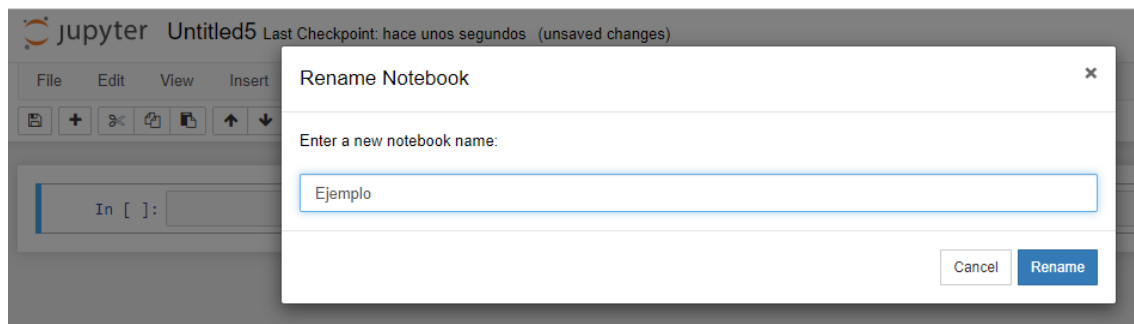


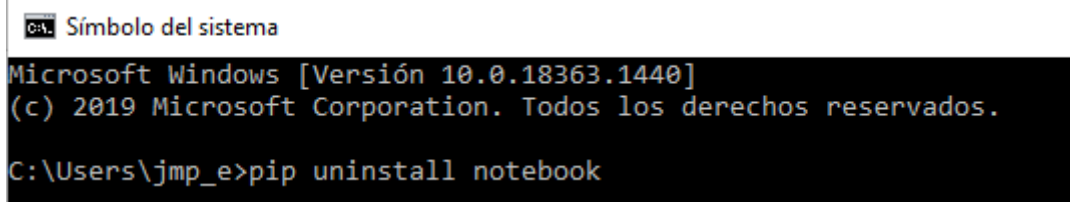
Figura 1.7: Renombrar el proyecto en Entorno Jupyter

El uso que haremos será muy simple.

- | Guardar (el disquete)
- | Añadir una celda (+)
- | Cortar una celda (las tijeras)
- | Ejecutar una celda (botón "Run"), etc.
- | Logout (servirá para salir, junto con cerrar la "cmd")

Desinstalar Jupyter

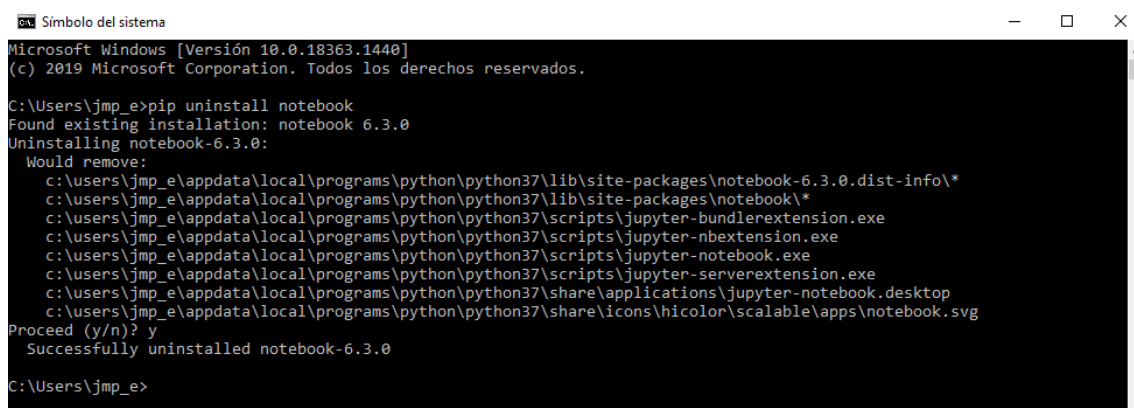
Y si queremos desinstalar el entorno "pip uninstall notebook"



```
C:\Users\jmp_e>pip uninstall notebook
```

Figura 1.8: Desinstalar el Entorno Jupyter

Nos preguntará si queremos proceder, continuar: y = yes, y lo desinstala.



```
C:\Users\jmp_e>pip uninstall notebook
Found existing installation: notebook 6.3.0
Uninstalling notebook-6.3.0:
  Would remove:
    c:\users\jmp_e\appdata\local\programs\python\python37\lib\site-packages\notebook-6.3.0.dist-info\*
    c:\users\jmp_e\appdata\local\programs\python\python37\lib\site-packages\notebook\*
    c:\users\jmp_e\appdata\local\programs\python\python37\scripts\jupyter-bundlerextension.exe
    c:\users\jmp_e\appdata\local\programs\python\python37\scripts\jupyter-nbextension.exe
    c:\users\jmp_e\appdata\local\programs\python\python37\scripts\jupyter-notebook.exe
    c:\users\jmp_e\appdata\local\programs\python\python37\scripts\jupyter-serverextension.exe
    c:\users\jmp_e\appdata\local\programs\python\python37\share\applications\jupyter-notebook.desktop
    c:\users\jmp_e\appdata\local\programs\python\python37\share\icons\hicolor\scalable\apps\notebook.svg
Proceed (y/n)? y
Successfully uninstalled notebook-6.3.0
C:\Users\jmp_e>
```

Figura 1.9: Entorno Jupyter desinstalado

2. PYTHON: RESUMEN EXPRESS

Para hacer un repaso intensivo del Lenguaje de Programación Python, y, tal vez, aprender cosas nuevas estaremos usando el Entorno de Desarrollo Jupyter.

La peculiaridad se encuentra a la hora de reconocer la extensión (.pynb) y no (.py)

Aunque habría formas de modificarlo, esto no será objetivo de esta asignatura.

Al mismo tiempo, podemos utilizar este entorno en Frameworks como Dash, por lo cual nos sirve para ambas cosas y es interesante conocerlo.

Si hubiera alguna cosa nueva, mejor que mejor. Python es muy amplio y siempre se pueden aprender más cosas.

Para la redacción de estos apuntes concretos estaremos realizando pantallazos del código con sus soluciones en el entorno. (In para el código Out la solución del mismo).

(Por ello esperemos que el/la estudiante no se despiste por el orden seguido en los números, que, por otra parte, serán correlativos en cada caso).

Comencemos!

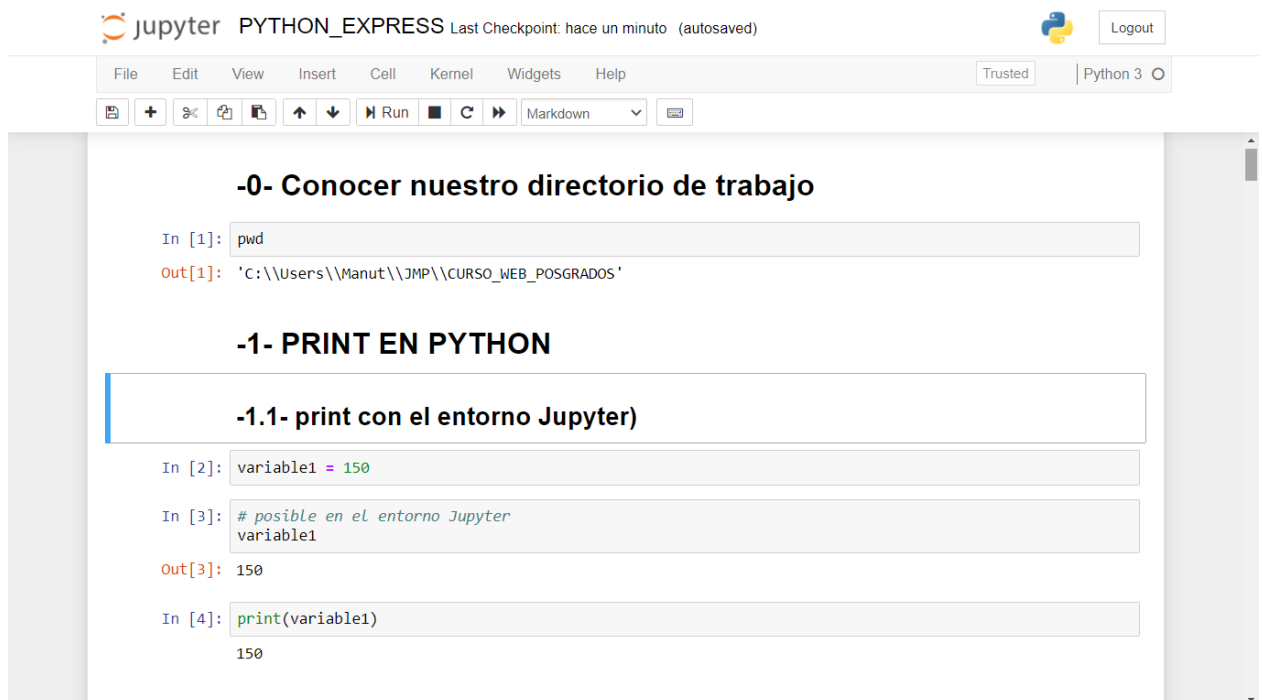


Figura 2.1: Ejemplo de Entorno Jupyter

El entorno queda de esta manera, donde podemos escribir código en las celdas, y podemos ver ejecuciones parciales de código, por tanto.

Escribiremos nuevamente nuestro código para que se pueda leer correctamente sin esfuerzo de vista.

Los pantallazos estarán seguidos en el orden en el cual se redactó el código.

Las soluciones a cada código (de cada celda) se mostrarán en el mismo lugar

De modo que Out[1] será resultado de In[1] y así sucesivamente.

Así mismo veremos el número de celda en el orden que se ejecutan.

-0- Conocer nuestro directorio de trabajo

```
In [1]: pwd
Out[1]: 'C:\\Users\\Manut\\JMP\\CURSO_WEB_POSGRADOS'
```

-1- PRINT EN PYTHON

-1.1- print con el entorno Jupyter)

```
In [2]: variable1 = 150
In [3]: # posible en el entorno Jupyter
        variable1
Out[3]: 150
In [4]: print(variable1)
150
```

Figura 2.2: Variables en Jupyter

-1.2- FORMAS DE IMPRIMIR EN PYTHON

```
In [5]: x = 2
        z = 4
        y = x + z

        print("la suma de:",x,"y",z, "es:", y)

        # otras formas de printear

        # ALTERNATIVA 1
        print("la suma de: " + str(x)+ " y " + str(z) + " es: " + str(y))

        # ALTERNATIVA 2
        print(f"la suma de: {x} y {z} es: {y}")

        # ALTERNATIVA 3
        print("la suma de: %s y %s es: %s" %(x,z,y))

        la suma de: 2 y 4 es: 6
        la suma de: 2 y 4 es: 6
        la suma de: 2 y 4 es: 6
        la suma de: 2 y 4 es: 6
```

Figura 2.3: Impresión en Python

-2- Nomenclatura Abreviada con Python

```
In [6]: L = [100,200,300,400]
        L

Out[6]: [100, 200, 300, 400]

In [7]: for numero in L:
        if numero<250:
            print(numero)

        100
        200

In [8]: [numero for numero in L if numero<250]

Out[8]: [100, 200]
```

Figura 2.4: Nomenclatura abreviada de Python

-3- Estructuras de Datos básicas

-3.1- Tuplas

```
In [9]: A = (100,200,300,400) # Posiciones: 0,1,2 y 3  
A
```

```
Out[9]: (100, 200, 300, 400)
```

```
In [10]: A[0] # 100
```

```
Out[10]: 100
```

-3.2- Numpy Arrays

```
In [11]: import numpy as np
```

```
In [12]: B = np.array([100,200,300,400])  
B
```

```
Out[12]: array([100, 200, 300, 400])
```

```
In [13]: B[0]
```

```
Out[13]: 100
```

Figura 2.5: Ejemplo de Estructuras de Datos

-3.3- Listas

```
In [14]: # Forma 1  
C = list((10,20,30,40))  
C
```

```
Out[14]: [10, 20, 30, 40]
```

```
In [15]: C[0]
```

```
Out[15]: 10
```

```
In [16]: # Forma 2  
D = [10,20,30,40,50]  
D
```

```
Out[16]: [10, 20, 30, 40, 50]
```

```
In [17]: D[0]
```

```
Out[17]: 10
```

Figura 2.6: Ejemplo de Listas en Python

-3.4- Mínimos y Máximos en estructuras de datos

```
In [18]: listado = [30,10,70,40]
         listado
```

```
Out[18]: [30, 10, 70, 40]
```

```
In [19]: min(listado), max(listado)
```

```
Out[19]: (10, 70)
```

Figura 2.7: Mínimos y Máximos

-3.5- Recomendaciones

```
In [20]: L = [100,200,300,400,500,600,700,800,900]
         # Correcto
         L
```

```
Out[20]: [100, 200, 300, 400, 500, 600, 700, 800, 900]
```

```
In [21]: # Pero mejor de esta forma..
```

```
In [22]: L = [
         100,200,300,
         400,500,600,
         700,800,900
         ]
         # Mejor
         L
```

```
Out[22]: [100, 200, 300, 400, 500, 600, 700, 800, 900]
```

Figura 2.8: Ejemplo de buenas prácticas

-3.6- Matrices con Numpy

```
In [23]: a = np.array([[10,20,30,40],
                        [50,60,67,80],
                        [90,45,23,18],
                        [10,50,26,93]])
a
Out[23]: array([[10, 20, 30, 40],
                [50, 60, 67, 80],
                [90, 45, 23, 18],
                [10, 50, 26, 93]])
```

Figura 2.9: Ejemplo de Matrices

-3.7- DataFrames

```
In [24]: # Usamos comillas dobles para los nombres de los estudiantes (E).
E = ["Pedro", "Ana", "Ricardo", "Luisa"]
E
Out[24]: ['Pedro', 'Ana', 'Ricardo', 'Luisa']

In [25]: # Notas de los exámenes (N) de 0 a 10, siendo 10 la mas alta
N = [10, 5, 8, 7]
N
Out[25]: [10, 5, 8, 7]
```

Figura 2.10: Ejemplo de DataFrames

```
In [26]: # pre instalado con Anaconda
import pandas as pd

# Creamos un dataframe
df = pd.DataFrame(E, columns = ["Estudiante"])
df

# Nombre de la columna = Nombre
```

Out[26]:

| | Estudiante |
|---|------------|
| 0 | Pedro |
| 1 | Ana |
| 2 | Ricardo |
| 3 | Luisa |

Figura 2.11: La primera columna del Dataframe

```
In [27]: # Añadimos una nueva columna.
# "Edad", valoress: Los valores de Q
df["Edad"] = N
df
```

Out[27]:

| | Estudiante | Edad |
|---|------------|------|
| 0 | Pedro | 10 |
| 1 | Ana | 5 |
| 2 | Ricardo | 8 |
| 3 | Luisa | 7 |

```
In [28]: # otra forma
```

```
In [29]: # Y todo de un paso con un diccionario
data = list(zip(E,N))
data
```

Out[29]: [('Pedro', 10), ('Ana', 5), ('Ricardo', 8), ('Luisa', 7)]

Figura 2.12: Dataframe completo

```
In [30]: import pandas as pd
df2 = pd.DataFrame(data, columns=["Nombre", "Edad"])
df2
```

```
Out[30]:
```

| | Nombre | Edad |
|---|---------|------|
| 0 | Pedro | 10 |
| 1 | Ana | 5 |
| 2 | Ricardo | 8 |
| 3 | Luisa | 7 |

```
In [31]: # e incluso..
```

```
In [32]: df3 = pd.DataFrame({"Nombre": E, "Calificación": N})
df3
```

```
Out[32]:
```

| | Nombre | Calificación |
|---|---------|--------------|
| 0 | Pedro | 10 |
| 1 | Ana | 5 |
| 2 | Ricardo | 8 |
| 3 | Luisa | 7 |

Figura 2.13: Uso de Diccionarios para los df

-3.8- Strings

```
In [33]: # el indice del string empieza en 0
s1 = "Hola, ¿cómo estás?"
s1
```

```
Out[33]: 'Hola, ¿cómo estás?'
```

```
In [34]: s1[0] # H
```

```
Out[34]: 'H'
```

```
In [35]: len(s1)
```

```
Out[35]: 18
```

Figura 2.14: Ejemplo de String

-4- Funciones, Clases y Objetos

-4.1- Concepto de Función

```
In [36]: def funcion_suma(x):  
        y = x + 10  
        return y
```

```
In [37]: funcion_suma(5)
```

```
Out[37]: 15
```

Figura 2.15: Ejemplo de Función

-4.2- Variables Globales y Locales

```
In [38]: x = 10  
        print(x)  
10
```

```
In [39]: def funcion_cambiar_x():  
        x = 5 # actúa SOLO aquí (MANERA LOCAL)  
        print(x)  
  
        funcion_cambiar_x()  
5
```

```
In [40]: print(x)  
10
```

```
In [41]: # Variables globales ahora
```

```
In [42]: x = 10  
        print(x)  
10
```

Figura 2.16: Ejemplo de Variables globales y locales

```
In [43]: def funcion_cambiar_x():  
        global x  
        x = 5  
        print(x)  
  
        funcion_cambiar_x()  
5
```

```
In [44]: print(x)  
5
```

Figura 2.17: Variable global

-4.3- Función lambda

```
In [45]: # con funciones
def funcion_suma_1(x):
    return x+5

# Llamada a la función
funcion_suma_1(10)
```

Out[45]: 15

```
In [46]: # Con Lambda
(lambda x : x+5)(10)
```

Out[46]: 15

Figura 2.18: Ejemplo de Función Lambda

-4.4- break, (continue y pass NO EXPLICADOS)

```
In [47]: L = [10, 20, 30, 40, 50, 60, 70]
L
```

Out[47]: [10, 20, 30, 40, 50, 60, 70]

```
In [48]: for numero in L: # 10-20-30-40-50-60-70
    if numero == 40:
        print("\n")
        break # cuando lo encuentra salta del bucle for
    else:
        print(numero) #

print("hemos llegado al 40, y salió del bucle FOR")
print("(este es el siguiente punto de ejecución)")

# 10-20-30 LES IMPRIME
# 40 SALTA (NO IMPRIME)
# 50-60-70 NO SE IMPRIMEN TAMPOCO
```

10
20
30

hemos llegado al 40, y salió del bucle FOR
(este es el siguiente punto de ejecución)

Figura 2.19: Ejemplo de break

-4.5- Clases (sin explicar herencia, etc)

```
In [49]: class Python:
          def imprimir(y):
              print("estamos aqui en la funcion imprimir ", y)

          def funcion_suma():
              x = 2
              z = 4
              y = x+z

              print("Estamos en funcion_suma y la suma de:",x," + ",z, "es:", y)

In [50]: Python.imprimir(5)

estamos aqui en la funcion imprimir 5

In [51]: # aqui no llamo en ningun momento a "imprimir"
          Python.funcion_suma()

Estamos en funcion_suma y la suma de: 2 + 4 es: 6
```

Figura 2.20: Ejemplo de entorno Jupyter

-4.6- POO

```
In [52]: # NOTA: init tiene 2 x barra baja (__)

          class Cliente:
              # funciones se les llama MÉTODOS
              # variables se les llama ATRIBUTOS
              def __init__(self, ID, Nombre, Edad, Estado_Civil, Profesion):
                  self.ID = ID
                  self.Nombre = Nombre
                  self.Edad = Edad
                  self.Estado_Civil = Estado_Civil
                  self.Profesion = Profesion

              # Instancio
              # genero tantos clientes como quiera de esta forma
              # cliente1 es el objeto1
              cliente1 = Cliente(1, "Ana", 30, "soltera", "ingeniera")
              cliente2 = Cliente(2, "Pedro", 45, "casado", "arquitecto")
              cliente3 = Cliente(3, "Andrea", 25, "soltera", "abogada")

In [53]: cliente1.Edad

Out[53]: 30
```

Figura 2.21: Ejemplo simple de POO

-4.7- try except

```
In [54]: x = [10,20,30,40]  
x
```

```
Out[54]: [10, 20, 30, 40]
```

```
In [55]: # print(w)
```

```
In [56]: # variable NO nombrada, ya no devuelve un error  
try:  
    print(w)  
except:  
    print("W no está definida, no podemos imprimirla")
```

W no está definida, no podemos imprimirla

```
In [57]: # variable nombrada (x)  
try:  
    print(x)  
except:  
    print("W no está definida, no podemos imprimirla")
```

[10, 20, 30, 40]

Figura 2.22: Ejemplo de try except

3. MUY BREVE INTRODUCCIÓN AL MACHINE LEARNING

El foco de la asignatura actual es la creación de Aplicaciones con Python, pero la utilidad REAL de crear estas aplicaciones es poder usarlas para nuestros proyectos de Machine Learning e Inteligencia Artificial.

El presente punto no pretende ser un manual de referencia para el aprendizaje de Machine Learning, sino ofrecer una explicación rápida y sencilla de algunos conceptos con el objetivo, con la idea, de que el/la alumno/a pueda entender cómo se hacen Aplicaciones para el Análisis de Datos con Machine Learning como finalidad.

Trataremos de dar respuesta -aunque breve- a preguntas como:

- | ¿Qué es un Dataset?
- | ¿Qué es X_train / y_train / X_test / y_test?
- | ¿Cómo se entrena un modelo de datos?
- | ¿Cómo se realiza una predicción en Machine Learning?
- | ¿Qué es Matplotlib?

Es imposible literalmente explicar al detalle estos conocimientos, y, para la presente materia serán No Evaluables o parcialmente evaluables, pero es conveniente explicarlo, dado que haremos pequeños ejemplos de aplicaciones que podrían ser escalables, y, que, si no entendemos un poco qué es cada cosa podría resultar abrumador.

Nuevamente mencionar, que aprender Machine Learning no es objetivo de esta materia, y de hacerse experto en la misma ya habrá tiempo, pero sí que necesitaremos tener una ligera idea, para que, cuando lleguemos a la parte de los Frameworks no nos resulte difícil de comprender lo que hacemos.

Para explicar esta parte hemos creado un nuevo ".ipynb", para la lección de hoy que aunque densa, es bastante introductoria, y que busca despertar el interés en el estudiante por la Ciencia de Datos.

De igual manera explicaremos esto, con pantallazos del propio .pynb de Jupyter.

De modo que, comencemos!

IRIS DATASET: ¿Qué es?

Es un set de datos muy típico en Machine Learning para principiantes

-1- Obtención de Datos:

Para hacerlo simple, lo haremos de la siguiente manera indicada. Existen mas formas de obtención de datos,

- mediante scrapeo web, ó, incluso,
- conectándonos a las APIs,

pero no será materia de este curso.

Símplemente hemos querido explicar esto para:

- Entender las posibilidades que tenemos si combinamos conocimiento
- Comprender los ejercicios realizados en Frameworks como Streamlit, Dash, etc.

Siendo, ésta, una asignatura de "**Aplicaciones Python**" y no de Machine Learning.

Pero necesitamos hacer cosas prácticas, osea, **Aplicaciones Python con Machine Learning**

Figura 3.1: Explicación inicial

Este dataset se basa en un conjunto de datos que contiene 50 muestras de cada una de tres especies de Iris (Iris setosa, Iris virgínica e Iris versicolor).

Para el estudio se midieron cuatro rasgos de cada muestra: el largo y ancho tanto para el sépalo como para el pétalo, (todo ello en centímetros).

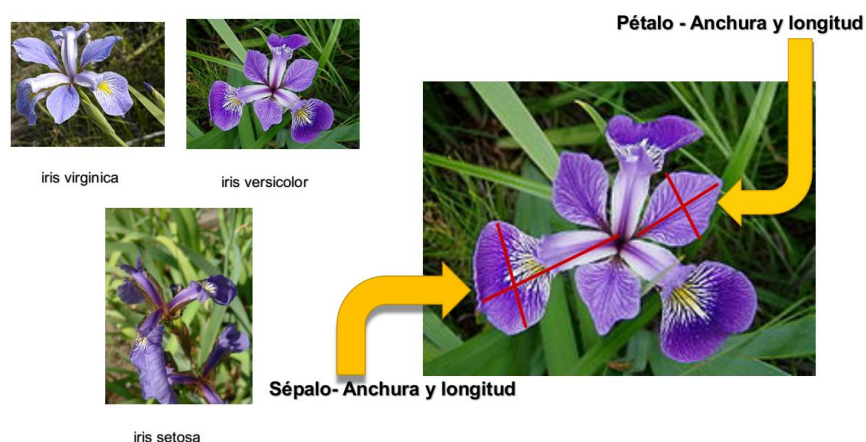


Figura 3.2: Pétalo y sépalo

-1.1.- Una posible forma

```
In [1]: # Nos vamos a este Link:
# https://archive.ics.uci.edu/ml/datasets/iris

# Download: Data Folder, Data Set Description
# Nos iríamos a: Data Folder
# y ahí obtenemos la data.
```

```
In [2]: """
INFORMACIÓN OBTENIDA DEL LINK ANTERIOR

Attribute Information:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica
"""
```

Figura 3.3: Información de las columnas

-1.2.- Otra posible forma de obtener los datos

```
In [3]: # 2ª forma de obtener el dataset:
# http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris
```

```
In [4]: from sklearn import datasets

iris = datasets.load_iris()
iris
```

```
Out[4]: {'data': array([[5.1, 3.5, 1.4, 0.2],
[4.9, 3. , 1.4, 0.2],
[4.7, 3.2, 1.3, 0.2],
[4.6, 3.1, 1.5, 0.2],
[5. , 3.6, 1.4, 0.2],
[5.4, 3.9, 1.7, 0.4],
[4.6, 3.4, 1.4, 0.3],
[5. , 3.4, 1.5, 0.2],
[4.4, 2.9, 1.4, 0.2],
[4.9, 3.1, 1.5, 0.1],
[5.4, 3.7, 1.5, 0.2],
[4.8, 3.4, 1.6, 0.2],
[4.8, 3. , 1.4, 0.1],
[4.3, 3. , 1.1, 0.1],
[5.8, 4. , 1.2, 0.2],
[5.7, 4.4, 1.5, 0.4],
[5.4, 3.9, 1.3, 0.4],
[5.1, 3.5, 1.4, 0.3],
[5.7, 3.8, 1.7, 0.3],
```

Figura 3.4: Obtención desde sklearn de Iris Dataset

```
Out[5]: array([[5.1, 3.5, 1.4, 0.2],
               [4.9, 3. , 1.4, 0.2],
               [4.7, 3.2, 1.3, 0.2],
               [4.6, 3.1, 1.5, 0.2],
               [5. , 3.6, 1.4, 0.2],
               [5.4, 3.9, 1.7, 0.4],
               [4.6, 3.4, 1.4, 0.3],
               [5. , 3.4, 1.5, 0.2],
               [4.4, 2.9, 1.4, 0.2],
               [4.9, 3.1, 1.5, 0.1],
               [5.4, 3.7, 1.5, 0.2],
               [4.8, 3.4, 1.6, 0.2],
               [4.8, 3. , 1.4, 0.1],
               [4.3, 3. , 1.1, 0.1],
               [5.8, 4. , 1.2, 0.2],
               [5.7, 4.4, 1.5, 0.4],
               [5.4, 3.9, 1.3, 0.4],
               [5.1, 3.5, 1.4, 0.3],
               [5.7, 3.8, 1.7, 0.3]])
```

[illegible]

Figura 3.5: Iris target

Para nuestro ejemplo proporcionaremos el .csv directamente

```
In [7]: # La forma que usaremos será la obtención de datos desde el CSV
```

-2- Importamos algunas dependencias generales

```
In [8]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Figura 3.6: Importamos dependencias principales

-3- Hacemos uso de pandas para importar la información .csv

Para ello tenemos en cuenta las siguientes aclaraciones

```
In [9]: # PEGAMOS LA RUTA DONDE ESTÁ ESTE .CSV EN NUESTRO PC
# # C:\Users\Manut\ALL\datasets\DATA
# y USAMOS "/" EN VEZ DE "\"

# en nuestro caso:
# C:/Users/Manut/ALL/datasets/DATA (ruta)
# 2_IrisSpecies.csv (nombre del documento con extensión .csv)

# df (nombre que damos)
# .head() nos permite imprimir las primeras 5 líneas

In [10]: df = pd.read_csv("C:/Users/Manut/ALL/datasets/DATA/2_IrisSpecies.csv")
df.head()
```

```
Out[10]:
```

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|----|---------------|--------------|---------------|--------------|-------------|
| 0 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Figura 3.7: Imprimo las primeras 5 filas

-4- Ploteo algunas gráficas

Usando Seaborn y matplotlib

```
In [11]: # https://seaborn.pydata.org/installing.html
# !pip install seaborn

In [12]: import seaborn as sns
```

Figura 3.8: Seaborn para hacer gráficas


```
In [13]: # Para el pétalo
sns.FacetGrid(df, hue="Species", height=4.8) \
    .map(plt.scatter, "PetalLengthCm", "PetalWidthCm") \
    .add_legend()

plt.title("Species en función del largo y ancho del pétalo")
plt.show()
```

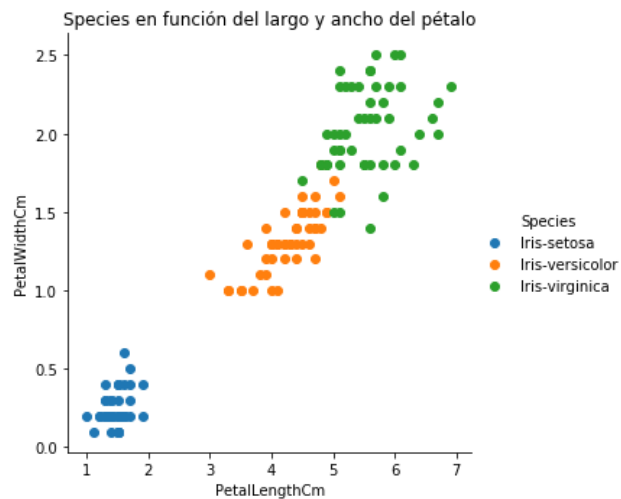


Figura 3.9: Gráfica para el pétalo

```
In [14]: # para el sépalo
sns.FacetGrid(df, hue="Species", height = 4.8) \
    .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
    .add_legend() \
    .set(title="Species en función del largo y ancho del sépalo")

# plt.title("Species en función del largo y ancho del sépalo")
plt.show()
```

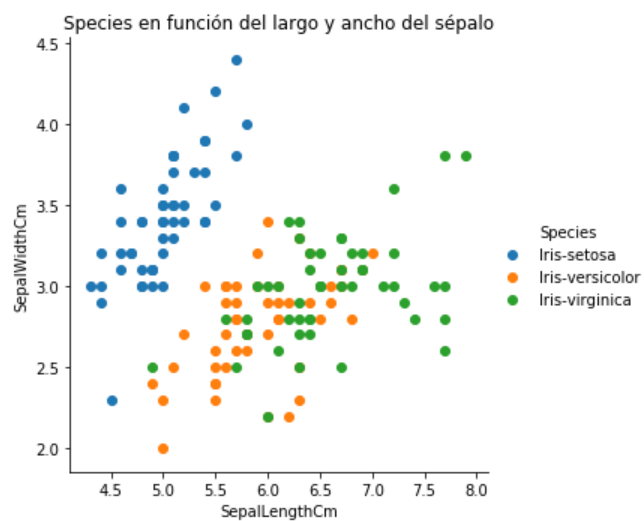


Figura 3.10: Gráfica para el sépalo

Es decir para el caso de las moradas:

- Para un largo de 4.3-5.8 y ancho mayor de 2.9 es generalmente de ese tipo
- Las demas es mas complejo de decidir

-5- Observo la información que tengo

y hago algunas transformaciones, si lo necesito

Generalmente uso `.value_counts()` para obtener información de la salida

Modifico si me interesa la información

EN ESTE CASO:

-Necesito transformar la salida en información para poder ser utilizada

```
In [15]: # Nombre de las columnas
df.columns

Out[15]: Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
               'Species'],
              dtype='object')
```

Figura 3.11: Nombres de las columnas

```
In [16]: # NombreDataFrame.NombreColumna.value_counts()
# Nos permite saber cuantos tipos diferentes de salida tenemos
# y su repetición en esa columna
df.Species.value_counts()

Out[16]: Iris-virginica      50
Iris-versicolor      50
Iris-setosa      50
Name: Species, dtype: int64

In [17]: # usamos .map() para transformar el texto en números
# existen mas formas de hacerlo.

# 'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
df.Species = df.Species.map({'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2 })
df.head()

Out[17]:
```

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|----|---------------|--------------|---------------|--------------|---------|
| 0 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 1 | 2 | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| 2 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| 3 | 4 | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| 4 | 5 | 5.0 | 3.6 | 1.4 | 0.2 | 0 |

Figura 3.12: Transformación texto a números

```
In [18]: # .tolist() me permite transformar esa información en una lista
print(np.array(df.Species.tolist()))

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Figura 3.13: Species

-6- Establezco X, y

```
In [19]: df.head(2)
```

```
Out[19]:
```

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|----|---------------|--------------|---------------|--------------|---------|
| 0 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| 1 | 2 | 4.9 | 3.0 | 1.4 | 0.2 | 0 |

```
In [20]: # drop me permite borrar, axis=1 le indico que las columnas.
X = df.drop(["Id", "Species"], axis = 1)
X.head()
```

```
Out[20]:
```

| | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm |
|---|---------------|--------------|---------------|--------------|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 |

Figura 3.14: Elimino información innecesaria

```
In [21]: # y, la salida, solamente la columna "Species"
y = df["Species"]
y
```

```
Out[21]: 0      0
1      0
2      0
3      0
4      0
..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

Figura 3.15: Asigno y

```
In [22]: # Algunas veces necesario para que no devuelvan errores posteriormente.
         X = X.values
         X
```

```
Out[22]: array([[5.1, 3.5, 1.4, 0.2],
 [4.9, 3. , 1.4, 0.2],
 [4.7, 3.2, 1.3, 0.2],
 [4.6, 3.1, 1.5, 0.2],
 [5. , 3.6, 1.4, 0.2],
 [5.4, 3.9, 1.7, 0.4],
 [4.6, 3.4, 1.4, 0.3],
 [5. , 3.4, 1.5, 0.2],
 [4.4, 2.9, 1.4, 0.2],
 [4.9, 3.1, 1.5, 0.1],
 [5.4, 3.7, 1.5, 0.2],
 [4.8, 3.4, 1.6, 0.2],
 [4.8, 3. , 1.4, 0.1],
 [4.3, 3. , 1.1, 0.1],
 [5.8, 4. , 1.2, 0.2],
 [5.7, 4.4, 1.5, 0.4],
 [5.4, 3.9, 1.3, 0.4],
 [5.1, 3.5, 1.4, 0.3],
 [5.7, 3.8, 1.7, 0.3],
 [5.1, 3.2, 1.5, 0.2]])
```

Figura 3.16: Asigno X

```
In [23]: y = y.values
y
```

[illegible]

Figura 3.17: y

-7- Divido mi set de datos en: train, test

(set de entrenamiento y set de test)

```
In [24]: from sklearn.model_selection import train_test_split
```

```
In [25]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
#                                     osea 75%-25%
```

-8- Utilizando algoritmos realizo una 1ª predicción

Lo 1º será decidir si estamos ante un problema de regresión, clasificación o clustering

en este caso, Clasificación Multivariable (salida 0-1-2)

Figura 3.18: División del Dataset en Entrenamiento y Prueba

DecisionTreeClassifier

```
In [26]: from sklearn.tree import DecisionTreeClassifier

In [27]: # 1-nuestro clasificador (clf) será DecisionTreeClassifier
clf = DecisionTreeClassifier()
# 2-Entreno con los datos que indico entre paréntesis.
clf.fit(X_train, y_train)
# 3-Realizo la predicción
y_pred = clf.predict(X_test)
# imprimo esa predicción
y_pred

Out[27]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
                0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2], dtype=int64)
```

Figura 3.19: DecisionTreeClassifier()

-9- Evalúo el modelo

```
In [28]: # Una posible forma para ello es accuracy_score
# existen muchas otras, pero por el momento es suficiente
from sklearn.metrics import accuracy_score

In [29]: # me dice que % de coincidencias hay
acc = accuracy_score(y_test, y_pred)
acc

Out[29]: 0.9736842105263158
```

0.9736 indica que el 97.36% de los datos han sido predecidos correctamente

Figura 3.20: Una posible forma para evaluar el modelo

RandomForestClassifier

```
In [30]: from sklearn.ensemble import RandomForestClassifier

In [31]: clf = RandomForestClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_pred

C:\Users\Manut\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning:
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

Out[31]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
                0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2], dtype=int64)

In [32]: acc = accuracy_score(y_test, y_pred)
acc

Out[32]: 0.9736842105263158
```

Figura 3.21: RandomForestClassifier()

***Nota**

Ignora ese warning, esto era lo que decía:

FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.

-11- Hacemos una NUEVA DIVISION de train y test

Es una de las posibles formas que tenemos de mejorar la predicción

```
In [33]: X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0, test_size=0.20)
#
```

DecisionTreeClassifier()

```
In [34]: clf = DecisionTreeClassifier()
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         y_pred

Out[34]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 2, 1,
                0, 0, 2, 0, 0, 1, 1, 0], dtype=int64)

In [35]: acc = accuracy_score(y_test, y_pred)
         acc

Out[35]: 1.0
```

Figura 3.22: Predicción para la siguiente división

En la práctica existen diferentes formas de mejorar la precisión de un modelo, y de hecho, habría que comprobar que los datos están "balanceados".

Dado que esta lección es introductoria, y el único objetivo es introducir el tema para que el/la alumno/a pueda crear alguna pequeña Aplicación con Machine Learning, por el momento será suficiente con tener medianamente claros los pasos seguidos.

RandomForestClassifier(n_estimators = 100)

```
In [36]: clf = RandomForestClassifier(n_estimators = 100)
         clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)
         acc = accuracy_score(y_test, y_pred)
         acc
```

```
Out[36]: 0.9666666666666667
```

-12- Haríamos la predicción que necesitamos

En la práctica, lo que solemos tener es un set datos sobre el que seguimos estos pasos

- Y que suelen llamar: "train.csv"

Y suele proporcionarse, a su vez un:

- "test.csv" sin información "de salida"

la cual deberemos predecir.

Probablemente es algo que se pueda aprender en otra Asignatura.

Figura 3.23: Usando el otro Algoritmo obtenemos otro resultado

Para poder ampliar más sobre este tema es recomendable esperar a otras lecciones, materias en las cuales seguro que se puede profundizar mucho más.

Es posible que en alguno de los siguientes temas hablemos de Series Temporales, obtención de datos de webs, e incluso sobre otras formas que tenemos de desarrollar este mismo código en menos tiempo, si cabe.

Por lo pronto esperamos que haya sido de ayuda, y que os despierte un mayor interés, si cabe, por la materia.

4. PUNTOS CLAVES

- | Dominar Python no será tarea simple puesto que es necesario conocer muy bien diferentes cosas, entre las cuales se encuentran las Estructuras De Datos. Las mismas nos van a permitir almacenar información y serán clave en la Ciencia de Datos.
- | El Entorno Jupyter nos permitirá hacer proyectos en los cuales podemos ir testeando el código a medida que lo vamos desarrollando, y el cual será, por tanto, muy buena elección para nuestros proyectos de Data Science.

